PATENT APPLICATION
Docket No. GE-0005US (RD29092)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:		)
		)
	Murren et al.	)
		)
Serial No:	09/847,067	)
		)
Confirmation No:	4549	)
		)
Filed:	April 30, 2001	)
		)
For:	Automatic Identification of Computer Program Attributes	)
		)
Examiner	Rachna Singh	)

The Honorable Commissioner of Patents
Mail Stop Appeal Brief - Patents
P.O. BOX 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

A Notice of Appeal was filed on August 17, 2007 including the appropriate fees. This Brief is being filed under the provisions of 37 CFR §41.37. The Filing Fee, including appropriate extensions under 37 CFR §1.136(a), as set forth in 37 C.F.R. §1.17(c), is submitted herewith.

TABLE OF CONTENTS

REAL PARTY IN INTEREST

The real party in interest is General Electric Capital, by way of assignment from Murren et al. who is the named inventive entity and is captioned in the present Brief.

RELATED APPEALS AND INTERFERENCES

While not formally related (as defined under 35 U.S.C.§§120 or 121), a Notice of Appeal has been filed in co-pending Application Serial Number 09/845,737 entitled; "Architecture and Process for Creating Software Applications for Multiple Domains," naming the inventive entity Murren et al. No other Appeals or Interferences are believed to be relevant under *M.P.E.P. §1205.02*.

STATUS OF CLAIMS

Allowed Claims:	None.
Canceled Claims:	Claims 3, 16, 20-29, 35 and 36 have been canceled.
Pending Claims:	Claims 1-2, 4-15, 17-19, 30-34, 37 and 38 are pending.
Appealed claims:	All of the pending claims are subject to this Appeal.

STATUS OF AMENDMENTS

A Final Office Action was issued on February 21, 2007.

A Notice and Appeal and a Response to the Final Office Action were filed in the U.S. Patent and Trademark Office on August 16, 2007. No claims were amended in that Response.

An Advisory Action maintaining the rejections issued on September 5, 2007.

SUMMARY OF THE CLAIMED SUBJECT MATTER

A testing and input form generation tool (hereinafter, "the input tool") and methods for automatically creating and identifying computer program attributes, are described in the patent application that is the subject of this Appeal. The input tool is used to automatically create code for form attributes to be input by a user or for presentation to the user in order to test a program including the code. The tool results in the automatic generation of a form having the attribute so the program can receive user data inputs or present data to the user.

The following is a brief summary of pending independent Claims 1, 11, 14 and 30 with references to the disclosure and figures, as required by 37 CFR §41.37(c)(v). However, the aforementioned references should not be interpreted as limiting any feature solely to the recited portions of the disclosure.

Claim 1 recites a method (FIG. 9, page 39, lines 10-14) comprising: accessing (page 30, lines 6-12; page 39, lines 10-18; FIG. 9, block 902) a computer program; automatically identifying (page 30, lines 8-9; page 37, lines 9-11; FIG. 9, block 906) a set of one or more attributes of the computer program with values that are to be input to the computer program by a user; and creating code (page 30, lines 6-8; page 35, lines 21-26; page 37, lines 21-25; page 38, lines 8-13; FIG. 9, block 916) for one or more forms (page 30, lines 4-5) including selected ones of the set of one or more attributes.

Claim 11 recites a method comprising: accessing (page 30, lines 6-12; page 39, lines 10-18; FIG. 9, block 906) a computer program, wherein the computer program includes a plurality of interactions (page 8, lines 15-20, page 39, lines 10-14) that each include one or more command definitions and one or more view definitions, wherein each command definition defines a command having various attributes and a behavior, and wherein each view definition defines a view that is a response to a request; and automatically identifying (page 30, lines 8-9; page 39, lines 15-18; FIG. 9, block 902) a set of one or more attributes of the computer program with values that are to be input to the computer program by a user wherein the automatically identifying comprises, identifying (page 31, lines 4-19; page 37, lines 9-11; FIG. 9, block 906), for each of the command definitions of each of the plurality of interactions, the methods of the command definition

(page 31, lines 20-21), checking, for each identified method that sets a value, whether a corresponding identified method obtains the value (page 34, line 19 - page 35, line 8), and identifying (FIG. 9, block 912), as an attribute of the set of one or more attributes, each attribute corresponding to a method that sets a value for the attribute for which there is no corresponding identified method (page 40, lines 1-16) that obtains the value for the attribute; and outputting an identification of the set of one or more attributes (914, page 40, lines 7-16).

Claim 14 recites a method comprising: accessing (page 30, lines 6-12; page 39 & lines 10-18; FIG. 10) a computer program; automatically identifying a set of one or more outputs of the computer program (page 35, lines 19-21); generating a list identifying the set of one or more outputs (page 35, line 24 - page 36, line 2); and outputting the list (page 36, lines 3-5 & lines 13-16), wherein the identifying and generating are performed based on an analysis of computer program code, independent of execution of the computer program to provide one or more views (page 36, line 21-page 37, line 8).

Claim 30 recites one or more computer-readable media comprising computer-executable instructions (FIG. 10) that, when executed, direct a processor to generate a plurality of input forms and output forms (page 30, lines 4-5) for a computer program by: accessing (page 30, lines 6-12; page 40, lines 10-18) the computer program to identify operations in the computer program (page 31, lines 7-10 & lines 24-27) that load attribute values and set attribute values; analyzing (page 33, lines 19-25; page 34, lines 4-7) the identified operations to determine one or more user inputs to the computer program; automatically generating code (page 30, lines 6-8; page 35, lines 21-26; page 37, lines 21-25; page 38, lines 8-13) for one or more input forms to allow a user to input at least some of the one or more user inputs; accessing the computer program to identify one or more outputs of the computer program (FIG. 9, block 912); and automatically generating code for one or more output forms to present the outputs of the computer program (FIG. 9, block 916).

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

1. Claims 37-38 stand rejected under 35 U.S.C. § 112, first paragraph as failing to comply with the written description requirement.

2. Claims 1, 2, 4-15, 17-19, and 30-34 were rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Published Patent Application No. 2004/0039993 to Kougiouris et al. (Hereinafter, "Kougiouris").

ARGUMENT

1. **FIRST GROUND OF REJECTION:** Appellant respectfully submit that the specification and Claims 37-38 are sufficiently written to allow one of ordinary skill to make and use the invention, and therefore satisfy the requirements of 35 U.S.C. § 112, first paragraph. Appellant respectfully requests that the 35 U.S.C. §112, paragraph one rejection be reversed.

The Final Action rejected Claims 37-38 under 35 U.S.C. §112, paragraph one (written description). Appellant responded by indicating that a written description commensurate with the subject matter of Claims 37-38 could be found, for instance, at page 30, lines 4-12; page 34, lines 8-18; and page 36, lines 21-24 of the specification. The present discussion is included in this Brief because the Advisory Action failed to indicate whether or not the rejection had been withdrawn.

As stated in M.P.E.P. §2163.03, there is a strong presumption in favor of there being sufficient written description existing in the specification. The Final Action has failed to indicate how the pending rejection is proper in light of this stated presumption. Rather, the Final Action attempts to disregard the burden of having to establish a *prima facie* case of non-compliance with the written description requirement, and thus attempts to shift the burden of proving compliance to the Appellant.

In particular, the Final Action fails to specifically identify how a person of ordinary skill in the art, reading the specification, including the above passages, would not be placed in possession of the subject matter. *M.P.E.P. §*2163.02. Rather, the Final Action merely notes that the "[s]pecification does not appear to discuss . . ." instead of 1) identifying passages in the text which appear to be relevant (or the closest) to the claimed subject matter and 2) then describe how particular passages of the Application indicate that the inventors were not in possession of the claimed subject matter. The Final Action provides no specific rationale as to why one of skill in the art would not understand the claimed subject matter based on the provided written description including the cited passages.

Further, Appellant respectfully submits that the Final Action does not provide any assertion which would cause one of ordinary skill in the art to question the inventor's possession of the claimed subject matter. In accordance with *M.P.E.P. §*2163.04, the burden is on the Office to provide 1) an initial rationale which indicates, under a

preponderance of the evidence standard, which limitations the Office believes are not adequately described and then 2) provide a *prima facie* case why, at the time of the invention, one of ordinary skill in the art would not have recognized that the inventors were in possession of the present subject matter.

In conclusion, the Final Action does not sufficiently articulate why one of ordinary skill in the art would doubt that the Application contains a sufficient written description as is required for a *prima facie* case of non-compliance with the written description requirement of 35 U.S.C. §112, paragraph one. Accordingly, it is respectfully submitted that Claims 37-38 satisfy the requirements of 35 U.S.C. § 112, paragraph one and therefore it is respectfully requested that the rejection of this rejection be reversed.

2. **SECOND GROUND OF REJECTION:** Appellant respectfully submits that Claims 1, 2, 4-15, 17-19, and 30-34 are not anticipated under 35 U.S.C. § 102(e) by U.S. Published Patent Application No. 2004/0039993 to Kougiouris et al. (hereinafter, "Kougiouris"). In particular, the Appellant submits that Kougiouris does not teach, or even suggest, "creating code for one or more forms including selected ones of the set of one or more attributes," as recited in Claim 1.

i. Kougiouris discloses a method for validating and formatting procedures for a graphical user interface (GUI) described in markup language.

Kougiouris is directed to a method of validating text and formatting text entered into a GUI text field. Kougiouris provides an example in which user text, in a social security entry box, in one instance, is entered as "555555555", and is validated, while an entry of "55y555555" is rejected because the text includes an alphabetic character "y", which is not permitted in a social security number. In the former example, the Kougiouris system automatically formats the text "555555555" as "555-55-5555", which corresponds to a properly formatted social security number. Kougiouris, paragraphs [0006] and [0010]-[0012].

Kougiouris discloses that this problem may be solved by including markup language attributes which automatically validate/format text for a GUI element. Kougiouris, paragraphs [0010]-[0012], reproduced directly below.

[0010] The problems outlined above may in large part be solved by providing a system and method for automatically performing validation and/or formatting procedures for a graphical user interface (GUI) described in a markup language file. The markup language may be any of various markup languages, including HTML and XML-derived markup languages. The graphical user interface markup language description may comprise descriptions of various types of graphical user interface elements for which text is to be validated/formatted, such as form fields, tables, hypertext links, etc.

[0011] An author of a markup language file may include various custom markup language attributes in order to automatically validate/format text for a GUI element. For example, an HTML file may include the following GUI element description:

<INPUT NAME="SSN" USFORMAT="usssn">

[0012] which includes a custom USFORMAT="usssn" attribute, indicating that the element should be validated/formatted according to a U.S. social security number pattern. Any of various codes or patterns may be supported in a particular embodiment, and an extensible framework enabling support for new types of codes or patterns to be easily "plugged in" may be utilized, as described below.

Kougiouris, paragraphs [0010]-[0012]

The cited portions of Kougiouris, in essence, disclose that the text being entered into the GUI field may be validated and formatted by a markup attribute so that the user entered contents is formatted and substantively conforms to a U.S. social security number. In other words, by using the attribute "usssn" the HTML file including the "usssn" element may 1) validate that the user entered text substantively conforms to a U.S. Social Security Administration number (i.e., nine digits and only numeric characters) and 2) the text is formatted as "xxx-xx-xxxx". Even more particularly, Kougiouris describes plugging in GUI elements, having markup attributes, when automatically instantiating the file. In Kougiouris, when presenting a GUI, ***an HTML file may be accessed*** and ***a text field is presented*** which validates user entered text substantively conforms to a U.S. social security number/formats user input as a social security number. A GUI element which does not have an attribute may accept all user entries.

Kougiouris discloses that a graphical display may be presented based on a markup language file which includes elements that have attributes. In other words, if presenting a GUI with a text entry box for accepting a social security number is desired, a markup element attribute "usssn," may be used, which will confirm that an entered number is valid and will further format a validated number so the number is arranged in a traditional format of three digits-dash-two digits-dash-four digits.

ii. Kougiouris does not disclose, either directly or inherently, "creating code" as contended by the Examiner.

Appellant respectfully disagrees with the rejection set forth in the Final Action. That is, Appellant submits that Kougiouris does not disclose each and every features as is required under 35 U.S.C. §102(e), and that the rejection is actually based on an incorrect assumption that "instantiating," as described in Kougiouris, and "creating code," as recited in Claim 1, are the same.

"Creating code" is recited in Claim 1 as, "creating code for one or more forms including selected ones of the set of one or more attributes." However, Kougiouris does not create code, as is claimed. Instead, Kougiouris discloses instantiating an already encoded markup file, as described in paragraphs [0070]-[0075] of Kougiouris, which are reproduced below.

[0070] In step 206, a user interacts with a GUI element for which validation/formatting information is specified in the markup language file. This user interaction may comprise any of various actions, such as changing the value of the GUI element, moving a mouse pointer over the GUI element, or other actions, such as described above with reference to FIG. 1. The user interaction of step 206 preferably causes the application program to generate a programmatic event associated with the GUI element, which the validation/formatting manager component is operable to receive in step 208. The interface between the application program and the manager component is discussed above.

[0071] As described above, the markup language file may comprise information regarding what types of validating/formatting operations the manager component should perform and what user interface events the manager component should respond to, and the manager component may be enabled to perform certain validating/formatting operations by default in response to receiving certain events. If the default configuration and/or the validation/formatting infor-

[0073] In one embodiment, the manager component passes the factory component the value of an attribute in the markup language file as an argument in order to identify the appropriate validation/formatting component. For example, for the above input form field description, the manager component may obtain the HSFORMAT="CCHcode" attribute information from the document object and pass the value "CCHcode" as an argument to the factory component, e.g. calling a "Get( )" method provided by the factory component, where this method returns a reference to an appropriate validation/formatting component instantiation.

[0074] In step 212, the validation/formatting manager component requests the validation/formatting component to perform a validation/formatting operation on the GUI element text. The manager component may obtain the GUI element text in any of various ways, e.g. by dynamically obtaining the text from the document object, by receiving the text as an argument along with the event in step 208, etc. The manager component may then request the validation/formatting component to perform the appropriate validation/formatting operation, as determined by the event received in step 208, the validation/formatting information specified in the GUI element markup language description, and the default configuration of the manager component. The manager component may perform this request by calling a method provided by the validation/formatting component, passing the GUI element text as an argument.

mation for the GUI element do not specify that the manager component should perform a validation/formatting operation for the event received in step 208, then the manager component may ignore the event.

[0072] In step 210, the validation/formatting manager component determines the validating/formatting component associated with the GUI element for which an event was received. As described above, the manager component may determine the validation/formatting information specified for each GUI element, e.g. by traversing a document object comprising this information. The manager component may use the validation/formatting information to determine the appropriate validation/formatting component for the particular text pattern or code associated with the GUI element. In one embodiment, a general framework enabling new validation/formatting components to be easily "plugged in" may be utilized. In this embodiment, the manager component may not have knowledge of particular codes or knowledge of how to instantiate particular validation/formatting components for these codes, but may instead simply call a factory component, where the factory component is responsible for instantiating the appropriate validating/formatting component.

[0073] In one embodiment, each validation/formatting component provides methods according to a standard validation/formatting interface that the manager component has knowledge of. This interface may define methods such as IsValid( ), for determining whether the text is valid text for the particular code, GetConforming( ), to get a properly formatted version of the text, etc. One embodiment of such a validation/formatting interface is described below. Thus, the validation/formatting manager component may to simply call the appropriate method of the validation/formatting component, regardless of the particular pattern or code associated with the GUI element.

Instantiating, referenced above, includes merely taking an element, ***which is already encoded in a markup language file***, and displaying the element as a GUI in accordance with the encoded instructions. That is, the elements in the markup file are already encoded so that a computer executing the file performs the instructions when displaying the GUI. The procedure described by Kougiouris procedure does not create code as is recited in Claim 1, but rather takes already encoded HTML elements and displays the element as a text box in a GUI.

For example, a SSN text entry box may confirm the entry is a nine digit, number only entry which is formatted as xxx-xx-xxxx. Thus, a markup element including an attribute is a set of encoded instructions which cause the computer to 1) display a GUI box for accepting text, 2) check to see if the text is nine digits, 3) check to see if the nine digits are only numbers, and, if so, 4) represent the nine digits in a three digit - two digit - four digit format. This procedure does not "create" code as recited in Claim 1 or "generate" code as recited in Claim 30. Rather, the process described by Kougiouris merely represents the element according to the markup file (in this case "usssn"). In other words, in Kougiouris, the code which causes the computer to display a GUI including the SSN entry box is already in existence. Therefore, code cannot be created but, instead, the markup element in Kougiouris merely "displays" or represents the markup element box according to the previously assigned attribute which were encoded in, for example, HTML.

The Final Action acknowledges that the element exists (*i.e.*, are encoded) because the Kougiouris markup file is a set of encoded instructions which cause the computer to

display the GUI described in the markup language file. As further indicated in the Advisory Action, the content "form" (*i.e.*, the GUI) is "displayed" rather than being "created" (Claim 1) or "generated" (Claim 30). Advisory Action, Page 2, excerpted first paragraph, reproduced below.

[0070]-[0075]. The graphical user interfaces can be created from markup languages such as HTML or XML-derived markup language descriptions. Markup languages such as XML or HTML provide the code for creating a graphical user interface. Applicant argues the act of displaying a form does not create the code for a form. Examiner disagrees because the form is displayed using code. See page 1, paragraphs [0010]-[0011]. Furthermore, Examiner disagrees that Kougiouris's presentations already exist and does not create the code for the forms because the forms accept input from a user and display the output based on the input. In other words, after input is accepted from a user, the HTML form is generated which requires the creation of code.

Advisory Action, page 2, first paragraph, excerpted.

Appellant respectfully submits that the rejection is untenable because, in Kougiouris, the coding exists as HTML or XML elements in the markup language file which displays the GUI. Therefore, in Kougiouris, code is not "created" when the markup file is displayed as a GUI. In order for the rejection to stand, the markup language file cannot be encoded as this would not disclose "creation", as recited in Claim 1, at the point of the computer displaying the GUI. The rejection is incorrect, and should be reversed, because the markup elements are encoded in the markup language file.

Therefore, for at least the foregoing reasons, the pending rejection of Claims 1, 2, 4-15, 17-19, and 30-34 under 35 U.S.C. § 102(e) has been traversed, and Appellant respectfully requests that the rejection, with regard to these claims, be reversed.

iii. Claims 2, 4-10 and 37 depend either directly or indirectly from Claim 1 and are allowable as depending from an allowable base claim, as well as for their own recited features.

Each of these claims depends from Claim 1 and is therefore allowable for at least the reasons set forth above with respect to Claim 1. These claims are also allowable for their own recited features which, in combination with those recited in Claim 1, are neither taught nor suggested in the references of record, either singly or in combination.

iv. Kougiouris discloses a method for validating and formatting user entered text. Kougiouris does not disclose whether a corresponding identified method obtains the value.

As discussed above in section (i), Kougiouris discloses a method in which user entered text is validated. In Kougiouris GUI element in a markup language file is written

with an attribute so that the GUI element checks to see if the user entered text which corresponds with text which is to be entered. What Kougiouris is validating is a user input. In contrast, independent Claim 11 recites, in part:

- checking, for each identified method that sets a value, whether a corresponding identified method obtains the value,
- identifying, as an attribute of the set of one or more attributes, each attribute corresponding to a method that sets a value for the attribute for which there is no corresponding identified method that obtains the value for the attribute, and
- outputting an identification of the set of one or more attributes.

In Claim 11, what is being "checked" is "each identified method" and not user input, as described by Kougiouris. More particularly, by Claim 11, the method may check an "identified method" that sets a value to determine if a corresponding method obtains the value. In contrast, Kougiouris merely discloses checking user input, which clearly is not an "identified method".

While the claim language is plain on its face, the detailed description of the present Application discloses this type of technique at page 32, line 25 through page 33, line 25. In particular, the written description states at page 33, line 19: "[b]y identifying matching or corresponding set and get methods, the test module 708 can determine which of the set methods in the interaction, if any, do not have values for their attributes loaded by a get method in that interaction."

The test module may identify matching or corresponding methods (page 33, lines 7-8) so that, a form, generated by a form generation module, may include a field to load the value (page 33, line 24-25). The written description notes that the test module may flag or identify conditions which may indicate whether the interaction is "well formed". Instant Application, page 34, lines 8-12. For example, "[t]he testing module 708 may optionally identify outputs of the interaction." Instant Application, page 35, lines 19-20. In contrast, Kougiouris is merely checks to see if user input matches acceptable input instead of checking to see if a "method obtains the value", e.g., is the method well formed. Accordingly, Claim 11 is clearly distinguishable from the teachings of Kougiouris, and therefore the corresponding rejection under 35 U.S.C. §102(e) should be reversed.

v. Claims 12 and 13 depend either directly or indirectly from Claim 11 and are allowable as depending from an allowable base claim, as well as for their own recited features.

Claims 12 and 13 depend from Claim 11 and are therefore allowable for at least the reasons set forth above with respect to Claim 11. These claims are also allowable for their own recited features which, in combination with those recited in Claim 11, are neither shown nor suggested in the references of record, either singularly or in combination.

vi. Kougiouris does not disclose, either directly or inherently, "identifying and generating are performed based on an analysis of computer program code, independent of execution of the computer program to provide one or more views" as contended by the Examiner.

The Final Action fails to establish a *prima facie* case of anticipation under 35 U.S.C. §102(e) with respect to, at least, the above features as recited in Claim 14. Accordingly, Appellant respectfully submits that the pending rejection should be reversed.

Appellant respectfully submits that Kougiouris does not disclose "identifying and generating are performed based on an analysis of computer program code, independent of execution of the computer program to provide one or more views." Appellant has previously referenced portions of the Application (page 31, lines 4-28) describing how the method recited in Claim 14 may be performed, in response to requests for clarification request. For reference, relevant portions of the Final Action (Response to Arguments) and the Advisory Action are reproduced below.

With respect to claim 14, Applicant argues there is no generation of a list identifying a set of one or more outputs and outputting the list. Examiner respectfully disagrees. Kougiouris discloses outputting the attributes in a form such as an HTML form in which the various attributes are listed. See figures 5A-5C. Outputting the attributes in a form is "outputting the list of outputs". The said identifying one or more outputs of a computer program is ***an analysis of the computer code***. Applicant argues this is not done independent of execution of the computer program. The claim recites "accessing a computer program; identifying a set of one or more outputs of the computer program". It is unclear how the output of a computer program can be analyzed without execution of the program. Clarification is requested.

Final Action Pages 10, last paragraph – page 11, first partial paragraph

With respect to claim 14, Applicant argues there is no generation of a list identifying a set of one or more outputs and outputting the list. Examiner respectfully disagrees. Kougiouris discloses outputting the attributes in a form such as an HTML form in which the various attributes are listed. See figures 5A-5C. Outputting the attributes in a form is "outputting the list of outputs". The said identifying one or more outputs of a computer program is an analysis of the computer code. Applicant argues this is not done independent of execution of the computer program. The claim recites "accessing a computer program; identifying a set of one or more outputs of the computer program". It is unclear how the output of a computer program can be analyzed without execution of the program. Clarification is requested.

Advisory Action, page 2, third paragraph.

Appellant respectfully submits that the position taken in the Final Action and the Advisory Action are in error because, Kougiouris only discloses "analyzing" the user input, which also requires that the GUI be executing as the text is "analyzed". In order to accept the position that Kougiouris could be used to analyze code, one would have to attempt to use the Kougiouris text validation feature in order to divine some information about the underlying code. This proposition is neither directly disclosed nor implied by the Kougiouris reference.

The Kougiouris technique simply does not analyze "computer program code" but instead, checks to see if user entered text is correct. As a result, Kougiouris neither directly nor indirectly discloses "an analysis of computer program code" as code itself is not being "analyzed" in the Kougiouris reference. In contrast, Claim 14 recites "analysis of computer program code."

Therefore, for at least the reasons set forth above, it is respectfully submitted that Claim 14 is distinguishable over Kougiouris, and that the corresponding rejection under 35 U.S.C. §102(e) should be reversed.

vii. Claims 15 and 17-19 depend either directly or indirectly from Claim 14 and are allowable as depending from an allowable base claim, as well as for their own recited features.

Each of these claims depends from Claim 14 and is therefore allowable for at least the reasons discussed above with respect to Claim 14. These claims are also allowable for their own recited features which, in combination with those recited in Claim 14, are neither shown nor suggested in the references of record, either singularly or in combination.

ix. The Final Action does not address each and every limitation of the claims and therefore has failed to make a ***prima facie*** case of anticipation. In particular the Final Action fails to specifically address the linguistic differences appearing in Independent Claim 30 in comparison to Claim 1.

As the Final Action has not specifically rejected the language recited in independent Claim 30 under 35 U.S.C. §102(e), Appellant requests the pending rejection be overturned based on the a similar rationale as discussed above with respect to Claim 1, although differences exist between Claims 1 and 30. In general, Kougiouris fails to disclose, "automatically generating code" or to "analyzing the identified operations". Kougiouris fails to do this because Kougiouris is directed to a method for displaying a GUI rather than generating code and analyzing identified operations as argued above. Accordingly, Claim 30 is patentably distinguishably over Kougiouris and the corresponding rejection under 35 U.S.C. §102(e) should be reversed.

x. Claims 31-34 and 38 depend either directly or indirectly from Claim 30 and are allowable as depending from an allowable base claim, as well as for their own recited features.

Each of these claims depends from Claim 30 and is therefore allowable for at least the reasons discussed above with respect to Claim 30. These claims are also allowable for their own recited features which, in combination with those recited in Claim 30, are neither shown nor suggested in the references of record, either singularly or in combination.

Conclusion

Appellant respectfully submits that the pending claims are distinguishable over the cited references of record. It is respectfully requested that the outstanding rejections be reversed, and that a Notice of Allowability be issued forthright.

Dated this_____day of_____, 2007

Respectfully Submitted,

Lee & Hayes, pllc

Dated: February 14, 2008

By: /David S. Lee – Reg. No.: 38,222/

David S. Lee
Reg. No. 38,222
Attorney for Appellant
(509) 324-9256

LEE & HAYES PLLC
1011 Western Ave.
Suite 906
Seattle WA, 98104
Telephone: (206) 315-7912
dslee@leehayes.com

CLAIMS APPENDIX

1. A method comprising:

accessing a computer program;

automatically identifying a set of one or more attributes of the computer program with values that are to be input to the computer program by a user; and

creating code for one or more forms including selected ones of the set of one or more attributes.

2. A method as recited in claim 1, further comprising generating a list including the set of one or more attributes and outputting the list.

3. (Canceled)

4. A method as recited in claim 1, wherein the one or more forms comprises one or more HyperText Markup Language (HTML) pages.

5. A method as recited in claim 1, wherein the one or more forms comprises one or more eXtensible Markup Language (XML) pages.

6. A method as recited in claim 1, wherein creating the one or more forms comprises generating a form definition for each form by:

identifying the selected one or more attributes to include on the form;

creating a data input field for the form definition via which a user can subsequently input a value for the attribute; and

creating a submit tag for the form definition via which the user can subsequently input a request to submit the values on the form to the computer program.

7. A method as recited in claim 1, wherein the computer program includes a plurality of interactions that each include one or more command definitions and one or more view definitions, wherein each command definition defines a command having various attributes and a behavior, wherein each view definition defines a view that is a

response to a request, and wherein the creating comprises creating one form for each of the plurality of interactions.

8. A method as recited in claim 1, further comprising:

identifying a set of one or more outputs of the computer program; and

outputting the set of one or more outputs.

9. A method as recited in claim 8, wherein outputting the set of one or more outputs comprises generating a list including the set of one or more outputs and outputting the list.

10. A method as recited in claim 8, wherein outputting the set of one or more outputs comprises creating one or more forms including selected ones of the set of one or more outputs.

11. A method comprising:

accessing a computer program, wherein the computer program includes a plurality of interactions that each include one or more command definitions and one or more view definitions, wherein each command definition defines a command having various attributes and a behavior, and wherein each view definition defines a view that is a response to a request; and

automatically identifying a set of one or more attributes of the computer program with values that are to be input to the computer program by a user wherein the automatically identifying comprises,

identifying, for each of the command definitions of each of the plurality of interactions, the methods of the command definition,

checking, for each identified method that sets a value, whether a corresponding identified method obtains the value, and

identifying, as an attribute of the set of one or more attributes, each attribute corresponding to a method that sets a value for the attribute for which there is no corresponding identified method that obtains the value for the attribute; and

outputting an identification of the set of one or more attributes.

12. A method as recited in claim 11, wherein identifying the methods of the command definition comprises querying the command definition to cause the command definition to identify its own methods.

13. A method as recited in claim 11, further comprising identifying one or more additional attributes that are not obtained by the computer program from elsewhere and that cannot be input by the user.

14. A method comprising:

accessing a computer program;

automatically identifying a set of one or more outputs of the computer program;

generating a list identifying the set of one or more outputs; and

outputting the list,

wherein the identifying and generating are performed based on an analysis of computer program code, independent of execution of the computer program to provide one or more views.

15. A method as recited in claim 14, further comprising:

automatically identifying a set of one or more attributes of the computer program with values that are to be input to the computer program by a user; and

outputting an identification of the set of one or more attributes.

16. (Canceled)

17. A method as recited in claim 14, further comprising creating one or more forms including selected ones of the set of one or more outputs.

18. A method as recited in claim 14, wherein the computer program includes a plurality of interactions that each include one or more view definitions, wherein each view definition defines a view that is a response to a request; and

wherein the automatically identifying comprises,

identifying, for each of the view definitions of each of the plurality of interactions, the methods of the view definition,

identifying, as one of the set of one or more outputs, each output included in one of the identified methods.

19. A method as recited in claim 14, wherein the computer program includes a plurality of interactions that each include one or more command definitions and one or more view definitions, wherein each command definition defines a command having various attributes and a behavior, wherein each view definition defines a view that is a response to a request, wherein the method further comprises creating a form for each of the plurality of interactions, and wherein each form includes one or more outputs for the corresponding one of the plurality of interactions.

20-29. (Canceled)

30. One or more computer-readable media comprising computer-executable instructions that, when executed, direct a processor to generate a plurality of input forms and output forms for a computer program by:

accessing the computer program to identify operations in the computer program that load attribute values and set attribute values;

analyzing the identified operations to determine one or more user inputs to the computer program;

automatically generating code for one or more input forms to allow a user to input at least some of the one or more user inputs;

accessing the computer program to identify one or more outputs of the computer program; and

automatically generating code for one or more output forms to present the outputs of the computer program.

31. One or more computer-readable media as recited in claim 30, wherein the computer-executable instructions further direct the processor to:

generate a list identifying the one or more user inputs to the computer program and another list identifying the one or more outputs of the computer program; and

output the lists.

32. One or more computer-readable media as recited in claim 30, wherein generating the one or more input forms comprises generating a form definition for each form by:

identifying the selected one or more user inputs to include on the form;

creating data input fields for the form definition via which a user can subsequently input a value for the user inputs; and

creating a submit tag for the form definition via which the user can subsequently input a request to submit the values on the form to the computer program.

33. One or more computer-readable media as recited in claim 30, wherein the computer program includes a plurality of interactions that each include one or more command definitions and one or more view definitions, wherein each command definition defines a command having various attributes and a behavior, and wherein each view definition defines a view that is a response to a request; and

wherein accessing the computer program to identify operations in the computer program that load attribute values and set attribute values, and analyzing the identified operations, comprises,

identifying, for each of the command definitions of each of the plurality of interactions, the methods of the command definition,

checking, for each identified method that sets a value, whether a corresponding identified method obtains the value, and

identifying, as one of the one or more user inputs, each attribute corresponding to a method that sets a value for the attribute for which there is no corresponding identified method that obtains the value for the attribute.

34. One or more computer-readable media as recited in claim 30, wherein the computer-executable instructions further direct the processor to identify one or more

attributes of the computer program that are not obtained by the computer program from elsewhere and that cannot be user inputs.

35-36. (Canceled).

37. A method as recited in claim 1, wherein the creating is performed in a form generation procedure in which said one or more forms are developed, the form generation procedure being independent and antecedent to a user's interaction with the computer program via said one or more forms.

38. One or more computer-readable media as recited in claim 30, wherein the automatically generating code for said one or more input forms and the automatically generating code for said one or more output forms are performed in a form generation procedure in which said one or more input forms and said one or more output forms are developed, the form generation procedure being independent and antecedent to a user's interaction with the computer program via said one or more input forms and said one or more output forms.

EVIDENCE APPENDIX

None.

RELATED PROCEEDINGS APPENDIX

None.